4

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER NW-LIS-89-09-11	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PHM - A Programmable Hardware Monitor		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER
AUTHOR(s) Craig S. Anderson, Katherine J. Armstrong Gaetano Borriello		8. CONTRACT OR GRANT NUMBER(s) N00014-88-K-0453
PERFORMING ORGANIZATION NAME AND ADDRESS Northwest Laboratory for Integrated Systems University of Washington Dept. of Comp. Science, FR-35 Seattle, WA 98195		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
CONTROLLING OFFICE NAME AND ADDRESS DARPA-ISTO 1400 Wilson Boulevard Arlington, VA 22209		12. REPORT DATE September 1989
		13. NUMBER OF PAGES 37
MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research - ONR Information Systems Program - Code 1513: CAF 800 North Quincy Street Arlington, VA 22217		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this report is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

NW-LIS, VLSI, CMOS, Chips, hardware monitor, programmable, custom CMOS.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes a project undertaken by the graduate students in CS 568 Winter 1989, to design and implement a programmable hardware monitor. The goal of this project was to create a general purpose, highly programmable yet compact design for a hardware monitor to be implemented with custom CMOS VLSI chips. The architecture is designed for easy extensibility and allows the monitor to be customized for use in a wide variety of computer systems, and to accommodate a broad range of applications. Although the project did not reach fabrication, a great deal of the layout was completed and simulated.

Proceedings of CS 568
PHM - A Programmable Hardware Monitor

Craig S. Anderson, Katherine J. Armstrong and Gaetano Borriello

Technical Report # 89-09-11

Department of Computer Science and Engineering, FR-35
University of Washington, Seattle, WA 98195 USA

August, 1989

Proceedings of CS 568
PHM – A Programmable Hardware Monitor

Craig S. Anderson, Katherine J. Armstrong and Gaetano Borriello

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

August 1989

Complementary metal oxide Semiconductor

Very Large Scale Integratn

Abstract

This technical report describes a project undertaken by the graduate students in CS 568, Winter 1989, to design and implement a programmable hardware monitor. The goal of this project was to create a general purpose, highly programmable yet compact design for a hardware monitor to be implemented with custom CMOS VLSI chips. The architecture is designed for easy extensibility and allows the monitor to be customized for use in a wide variety of computer systems, and to accommodate a broad range of applications. Although the project did not reach fabrication, a great deal of the layout was completed and simulated.

1 Introduction

The purpose of a hardware monitor is to detect and record hardware "events" in a computer system for the purposes of performance evaluation. A simple, and much used, approach to performance analysis is trace-driven simulation. Trace-driven simulation depends upon the collection of a trace of all addresses generated during the execution of one or more computer programs. The major problem of this approach is that a large amount of memory is required to store all the trace information. Additionally, methods for collecting address traces are often expensive and intrusive; the process of collecting the trace data may alter the behavior of the computer system while it is being traced. Keywords: layout, (KR)

Hardware monitoring is sometimes used as an alternative to full address tracing. A hardware monitor is attached to a set of wires, e.g. an address bus, and the monitor's internal counters are used to count the occurrences of events on those wires. Measuring cache performance by collecting cache hit statistics is a common example of the use of hardware monitoring techniques. The values of the monitor's counters are dumped to some storage medium, usually disk or tape, at regular intervals. Software programs may be run later, using the data collected as input. Hardware monitoring has been an attractive alternative to full address tracing, both because of the substantially reduced memory requirement, and because monitoring the hardware directly makes it possible to collect information, such as operating system activity, that may be difficult or impossible to collect in software. Unfortunately, this attractive filtering feature of hardware monitors has also been a major limitation on their usefulness. Merely counting events does not provide a very full picture of the activities of the computer system.

PHM *Programmable Hardware Monitor* was designed to provide a more flexible alternative. It is fully programmable and can be used efficiently both as a counter of events and as a mechanism for collecting filtered event traces. It is designed to operate at the same speed as the machine being monitored (up to 20 MHz), thus providing maximal throughput with minimal interference. The monitor board will consist primarily of two types of custom CMOS VLSI chips: data filter chips and data compression chips. The filter chip will be described in detail in this report. The reader is referred to [Bunton 89] for information on the data compression chip. In addition to the custom chips, there will be bus interface logic, which will allow the monitor to be programmed from a PC keyboard, and disk control logic, which will handle the movement of data from the data compression chips to disk storage.

The remainder of this paper is organized as follows. In section 2 we elaborate on the use of hardware monitors and provide a characterization of the desired functionality of a hardware monitor. Section 3 contains an overview of the architecture of PHM. The major features are outlined and the motivation for certain decisions is discussed. In section 4, the layout of the filter chip is detailed along with brief descriptions of the basic leaf cells. Finally, the current status of the project and future work are discussed in section 5.

2 Features and Uses

Traditionally, hardware monitors have enjoyed only a limited usefulness. A hardware monitor's primary advantage over address trace collection software lies in its ability to collect data relatively non-intrusively and at a high rate of throughput. However, to achieve these goals, hardware monitors generally impose a filtering requirement – they are only able to count events and save those counts. Many potential applications for hardware monitors require further capabilities. Examples of such applications include: collecting traces of some subset of possible addresses, recording cache misses and run lengths of cache hits, monitoring the time between procedure calls and returns, counting the number of instructions or amount of time between acquiring and releasing a lock. To be general enough to handle such a broad range of applications, we would like a hardware monitor to have the following capabilities:

- Detect the occurrence of any of a number of events on the set of wires being monitored;
- Count occurrences of those events and selectively output those counts;
- Collect a trace of the actual events of interest, rather than just a set of counts;
- Extract the values on all or a subset of the monitored wires;
- Take time into account, e.g. record events which occur at some fixed interval of time apart from each other, record the time between two events, or record a "timestamp" indicating the time at which an event occurred;
- "Remember" an event which occurred in the past, e.g. record only those occurrences of event A that occurred n cycles before an occurrence of event B.

We have incorporated all of these features into the PHM filter chip. In addition, the onboard data compression chips, which use a fast new variation of the Lempel-Ziv algorithm, greatly enhance the filtering capabilities of our hardware monitor and its ability to collect data at a very high throughput rate. In the next section, the architecture of the filter chip, which lies at the heart of our monitor design, will be detailed.

3 Architecture

3.1 Features

The data filter chip is composed of seven major functional elements: comparators, switches, event recorders, internal counter/timers, external counters, ID code logic, and programming logic. In addition to the data filter and data compression chips, a set of trace registers is included on the monitor board. The loading and saving of data in the trace registers is controlled by the filter chip. The filter chip takes as input a subset of the wires of interest. It monitors the signals on those wires to determine when to perform a tracing function. Having a variable number of trace registers located on the monitor board, rather than a fixed number internal to the filter chip, makes it possible for PHM to use the signals seen on one set of wires as indicators of when to save the pattern on a different set of wires. It also allows for collection of multiple data streams simultaneously. The monitor board will also include a time-of-day counter and, optionally, a multiplexor, which is used to select which of three data streams – monitored wires, filter chip counter values, and time-of-day – are to be saved on any given cycle.

By far the major portion of the filter chip is occupied by the programmable comparators. Each bit of each comparator may be separately programmed with a key and a mask. The key bits are programmed with a pattern which, when seen on the monitored wires, will produce a *comparator match*. Mask bits are added to allow the "don't care" condition to be specified for any bit. The comparator cells were designed such that both the number and width of the comparators may be easily extended, constrained only by the physical area of the chip and the target cycle time. Our current design has 64 60-bit comparators. The number of comparators was determined primarily by space considerations, while the width was dictated by the 50 ns target cycle time. The issues will be discussed further in section 4.

The second most area-consumptive element of our design is a set of programmable switches. The hardware realization for the switch cell is quite similar to that of a CAM (content-addressable memory) cell. These switches are programmed to select the actions to take place when a comparator match occurs. A separate set of switches is provided for each of the 64 comparators so that the actions resulting from a comparator match may be separately selected for each of the up to 64 possible Boolean conditions being monitored. The actions which may be selected include setting and resetting event recorders, incrementing or resetting a counter, recording a timestamp, loading an external trace register, or sending the value stored in a trace register to the data compression chips. An itemization of these control signals follows:

- COUNT, RESET_COUNTER, and OUTPUT_COUNTER for each of the external counters;
- COUNT and RESET_COUNTER for each of the internal counter/timers;

- SET and RESET for each of the event recorders;
- INC – a chip output indicating that the current counter output should be incremented. This "virtual increment" feature was added so that any counter can be incremented and output in the same cycle;
- LOAD – a chip output which signals that one or more external trace registers should be loaded;
- TRACE_EXT – a chip output that causes the content of one of the external trace registers to be sent to the data compression chips;
- DATASEL_EXT – a chip output indicating that the current values on the data wires should be saved (used with LOAD and TRACE_EXT);
- TIMERSEL_EXT – a chip output indicating that the current value of an external time-of-day counter should be saved (used with LOAD and TRACE_EXT);
- COUNTSEL_EXT – a chip output indicating that the current value of one of the filter chip's counters should be saved (used with LOAD and TRACE_EXT).

The latter three signals can be used to multiplex between the three types of data streams – monitored data wires, time-of-day, and filter chip counter values – choosing which of these streams is sent to the data compression chips in any given cycle. The LOAD and TRACE signals allow for the recording of past events. When the first event is seen, the filter chip issues a LOAD signal. A TRACE signal is issued only if a specified second event occurs at some point in the future.

In addition to the comparators and switches, there are eight event recorders, which are implemented as simple S-R flip-flops. An event recorder is used to record the occurrence of some Boolean condition on the monitored wires. This is accomplished by programming the switches such that when a comparator match occurs, a switch associated with that comparator issues a SET signal to a particular event recorder. The current values of all eight event recorders are cycled back to the comparators, allowing PHM to detect events composed of several Boolean conditions, perhaps separated by time. This is equivalent to a set of parallel state machines for the up to 456 states.

There are two sets of counters on the filter chip. One set consists of two eight-bit internal counter/timers, the outputs of which, as with the event recorders, are circulated back as additional inputs to the comparators. Using these additional inputs, the number of times a Boolean condition occurs or the number of cycles between two Boolean conditions can be included in the event definitions which are programmed into the comparators. The other set of counters on the filter chip is composed of eight 32-bit external counters. We call these counters "external" because their outputs are chip outputs which may be saved, compressed, and stored by the monitor. This is in contrast to the "internal" counter/timers described above, whose outputs are used internal to the chip, but are not seen outside the chip. As with the comparators, area and timing considerations were of primary importance in choosing the number and width of the two sets of counters. The constraint on the number of external counters derives from the necessity of providing three switches per comparator (to produce the COUNT, RESET_COUNTER, and OUTPUT_COUNTER control signals) for each additional counter. Because of the width of the comparators and switches, there is very little extra space in the horizontal dimension on the chip.

The filter chip is a pipelined architecture. Because our target cycle time is 50 ns, two full cycles are required to fully process each word of incoming data (see section 3.2). The counters, however, are slow. They are not updated to reflect their new state, resulting from the detection of any programmed event, until two cycles after the occurrence of that event. Since the comparator pipe-stage processes a new word of incoming data every cycle, it cannot wait two cycles for the counters. To get around this problem, the COUNT and RESET control signals, in addition to being sent to the internal counter/timers, are circulated back to the comparators from the switch outputs. These signals, in conjunction with the old counter/timer values, act as a substitute in the comparison phase for the as yet unavailable updated counter/timer values.

Included on the filter chip is a logic block which produces a code used to identify which comparators matched on any given cycle. Because of chip area and pin limitations, this logic does not produce a unique code for every possible combination of comparators. Rather, the code produced is a simple 6-bit pattern which is the logical OR of the comparator numbers of those comparators which generate a match. Since more than one comparator may generate a match in some cycles, we have added a 1 bit code enable register (CER) for each comparator. Only if the CER has been programmed to 1 will a match from the corresponding comparator cause that comparator's code to be OR-ed into the output code. It is the programmer's responsibility to make use of the CERs and manage the placement of conditions in comparators such that the comparator codes, when OR-ed together, are meaningful. While this solution is far from optimal, both because of the potential for code conflicts and because of the added programming complexity, we feel that the large number of comparator placement options available to the programmer make this tradeoff for space reasonable.

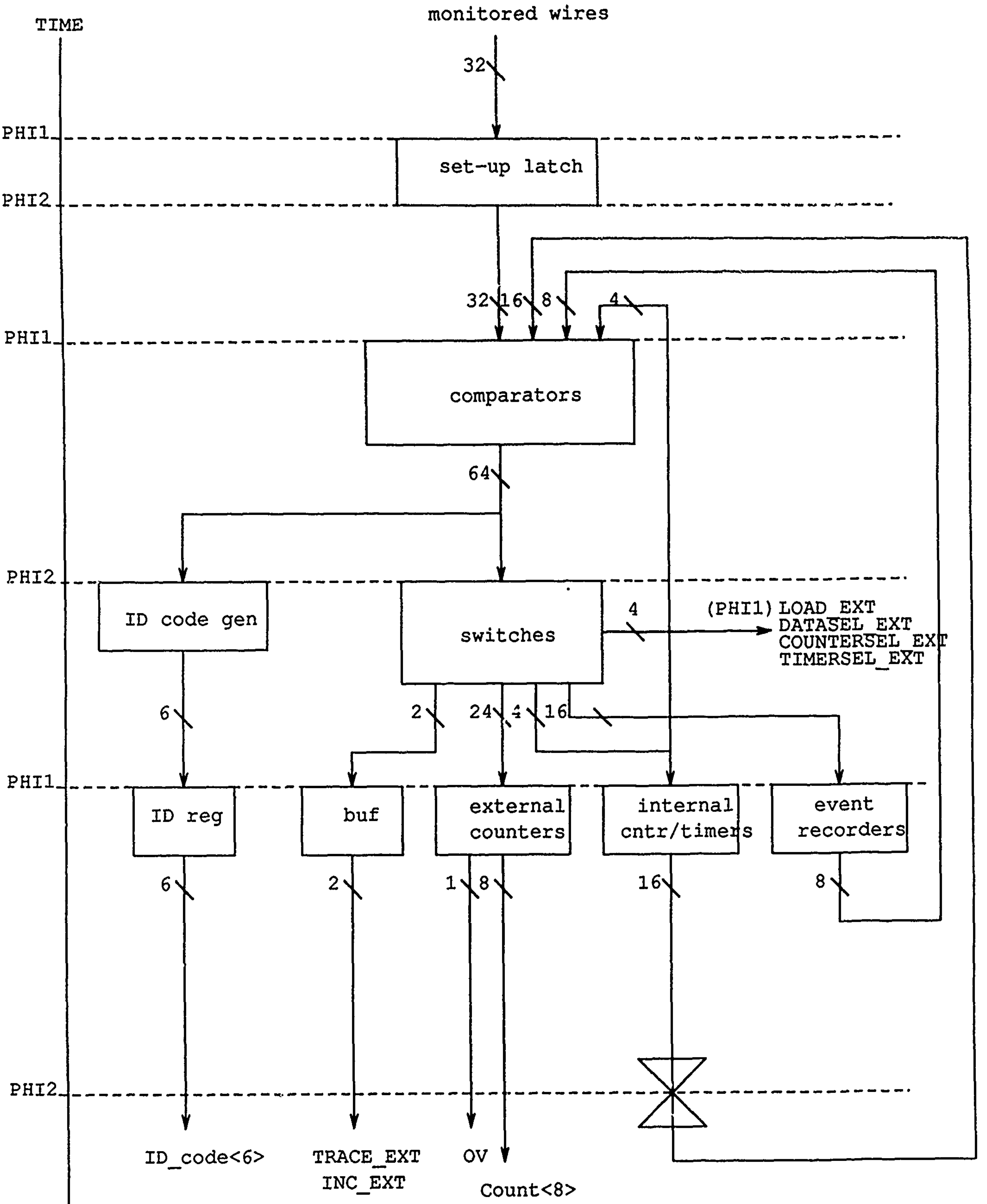
The programming logic, the last of the major functional blocks which make up the filter chip, is a distributed entity composed primarily of decoders. Twelve pins are used to control programming of the chip. This is in addition to the 32 data input pins which provide the data to be programmed into the chip (comparator keys and masks, switch masks, and CER values). The monitor board will include bus interface logic so that it can be programmed from a personal computer.

3.2 Timing

The filter chip was designed to accommodate a 50 ns cycle time, i.e. the monitored wires may be sampled and processed every 50 ns. To achieve this throughput rate, we employ a two-cycle pipelined architecture, using precharge logic between evaluation phases of a two-phase (phi1, phi2) non-overlapping clock. In addition to the two processing cycles, an initial set-up cycle is required to synchronize the input with the operation of the chip.

Figure 1 illustrates the timing of the flow of a word of data through the chip. For clarity in the following discussion, we will use c0 to refer to the cycle each word spends at the set-up latch, c1 to refer to the cycle spent at the comparators and switches, and c2 to refer to the cycle spent at the counters, event-recorders, etc.

At c1.phi1 inputs are passed from the set-up latch to the comparators, along with the values of the event recorders, internal counter/timers, and the four counter/timer control signals. The comparators evaluate and the comparator match line outputs are passed to the switches at c1.phi2. These values are compared with the mask bits programmed into the switches to determine which control signals should be asserted if a match occurs. Also at c1.phi2, the comparator match values



(NOTE: PHI1 and PHI2 along the left margin indicate the clock phase at which data passes from one stage to the next.)

FIGURE 1

are passed to the ID code generation block, where a code is generated indicating which comparators matched the input.

The ID code is latched into an ID register at c2.phi1, and is held there for output from the filter chip at c2.phi2. The switch outputs that control the external counters, internal counter/timers, and event recorders are also available at c2.phi1, as well as the LOAD_EXT, DATASEL_EXT, COUNTERSEL_EXT, and TIMERSEL_EXT signals, that control the external trace registers. These latter signals are immediately output from the filter chip. Pass gates are added between the internal counter/timers and the comparators to ensure that the updating of the counter/timers does not interfere with the evaluation of the comparators. The counters are all given the full cycle c2 to evaluate. The other switch outputs, the TRACE_EXT and INC_EXT signals, are gated into buffers at c2.phi1 and held for output on the next clock phase (c2.phi2). The use of INC_EXT will be explained in the next section. As an example of the use of the external trace register control signals, if we wish to determine the time between an event A and another event B, we simply assert LOAD_EXT and TIMERSEL_EXT when A is seen, and then again when B is seen. The difference between these times can be determined by software.

At c2.phi2, the filter chip outputs which are available are the signals TRACE_EXT and INC_EXT, the ID code, and any counter value which is to be traced. In addition, if any of the counters overflows, an OVERFLOW signal is output. No assumptions are made about how overflows are to be handled. In particular, the chip will continue to operate after an overflow, as if no overflow has occurred.

3.3 More Design Decisions

We were not able to find a way for the hardware to guarantee that no more than one external counter value would be output in a given cycle without using a lot of slow circuitry. We made the decision, therefore, that this guarantee must be made by the programming software. If more than one counter value needs to be output following some particular event, this can be handled by using different comparators and event recorders to sequence through the counters, outputting one at each cycle, e.g.

```
if (ER1 is set) then
    output counter2 and set ER2
if (ER1 and ER2 are set) then
    output counter5 and set ER3
...
```

(Setting ER1 indicates that it is time to output the counters.)

Resetting the counters is handled in the following manner: Instead of resetting a counter when its value is output, counters are "virtually reset" at the time when they are next used. This is accomplished by redefining the COUNT and RESET_COUNTER signals to COUNT_PAST_ONE and LOAD_ONE. The first time an event is seen which we want to count, we assert the LOAD_ONE signal, which causes the counter to be reset to 1. Thus, we are resetting and counting to one in one operation. After the first time an event is seen, all subsequent occurrences of the event cause the COUNT_PAST_ONE signal to be asserted, which is a regular increment signal. Thus the "reset" of a counter never occurs while that counter is being output. The cost is that one extra comparator is required each time a new event is to be counted. Instead of using one comparator to count every time an event is seen, we have one comparator to count the first time an event is seen, and one comparator to count all subsequent occurrences of that event.

In order to handle the case where we want to increment a counter and output its incremented value during the same cycle (for example, if we are counting the number of times instruction A is seen in 10 clock cycles, and we see instruction A on the 10th cycle), we have an extra switch output signal called INC_EXT. INC_EXT dces not change the counter value. Instead, this signal is sent out of the chip along with the unincremented counter value, and the "virtual incrementing" can be handled by software.

The filter chip layout is very tightly packed in the horizontal dimension. It is the width of the comparator and switch blocks which takes up most of this horizontal real estate. To try to alleviate some of the crowding, we considered a design involving two sets of switch cells. Inputs to the first set would be the comparator outputs; the outputs would be inputs to the other set of switches. The second set would generate as outputs the control signals for the counters, ERs, etc. This second set would be oriented perpendicular to the first set of switches and the comparators. Thus, using this design, we could have decreased the width of the chip, since fewer switch cells would have been required in the horizontal dimension. The amount of parallel activity (setting counters, ERs, etc.) that could be performed on each cycle is somewhat decreased in this design, subject to the width of the first set of switches. We did not adopt this architecture because adding a second set of switch cells would have added at least another 1/2 cycle to the datapath. Even with our internal feedback scheme, we would not have been able to process one input per cycle. A possible compromise might be to use two sets of switch cells only in cases where the control signals generated are not on the critical path, for example for the external counters. The tradeoff would be a little added complexity in return for a savings in horizontal real estate.

4 Layout

A great deal of the layout for the filter chip has already been done. The basic cells for the comparators, switches, counters, code generator, and event recorders have been completed. To reduce the amount of signal routing needed for the final chip, many of the cells were pitch-matched so that the connections between blocks could be made by abutment. In some cases, special cells called *glue cells* were constructed to make the various blocks fit together cleanly. Also, extensive use of precharge logic was made to reduce the number of logic gates needed for the chip, saving both space and time. The current size of the main block (which consists of the comparators, switches, programming decoders, and code generator) is 7200 microns wide by 3400 microns high. The main leaf cells in this block were designed under the supposition that the height would be the limiting factor in the chip layout; therefore, the cells were constructed to minimize height at the expense of width. Unfortunately, the supposition was wrong, leading to a layout constrained in the horizontal direction. The size given for the main block does not include space for the counters, the event recorders, or any routing that will need to be done. Simulations run on the leaf cells indicate that the goal of accommodating a 50 ns cycle time can be met. In the following sections a brief overview of each cell is presented.

4.1 Comparator Cell

Because of the large number of comparator cells called for by the architecture of the filter chip, minimizing the area of the comparator leaf cell was critical to our design. In addition, the comparison operation had to be fast to meet our overall goal of a 50 ns cycle time. To save space, precharge logic was used for the comparator match lines. By using precharge, we were able to use wired-*AND* logic for the rows of comparator cells, saving the space and delay that it would take to compute a 60 input *AND* function using logic gates. Once the key and mask registers in the comparator cell have been loaded, the match line is precharged during one phase of a two phase non-overlapping clock. During the second phase of the clock, the match line is conditionally pulled low, depending on the the value of the data input and the values of the comparator's key and mask registers. To reduce the time required to complete a comparison operation, and to achieve a more straightforward design, the block of 60 comparators was split into a block of 32 comparators whose inputs come from off-chip, and a bank of 28 comparators whose inputs come from the counters and event recorders. The match line was split between the two banks, making it necessary to *AND* the value of the two match lines to get the final result of the comparison. The reduced capacitances of the two shorter match line significantly reduced the worst case performance of the design. In the worst case, with only a single comparator cell pulling down a match line in a row of thirty-two comparator cells, the comparison operation requires 24 ns.

4.2 Switch Cell

The switch cell was laid out so that connections with comparator cells could be made by horizontal abutment. To save space, the switch cell is also pre-charged. The pre-charge is performed during the second phase of the two phase clock, and the switch cell evaluates on the first phase. The result of the comparison operation is generated during one active phase of the two-phase clock, and is to be used by the switches during the other active phase of the clock.

4.3 Counter

All of the counters in the filter chip are synchronous with a reset capability. To implement reset and count in one cycle, the counter was designed to reset to a value of 1, thus effecting a simultaneous reset and count.

4.4 Miscellaneous Cells

The remaining cells that have been constructed are uninteresting. The event recorders are simple S-R latches consisting of cross-coupled NOR gates. Other layout which has been completed includes the code generator cell, which is quite similar to the switch cell, decoders for programming the switches and comparators, and a multiplexor.

5 Current Status and Future Work

Though the leaf cell design and assembly of the main block are largely complete, much work remains to be done. The counters and event recorders have not been placed. Some of the layout for the glue logic, which will be needed to connect functional blocks, has not been completed. And finally, global routing and pin assignment have yet to be done. In addition, much simulation will need to be done as the layout is assembled to ensure that the chip meets functional and timing specifications.

In addition to the final assembly work that needs to be completed, two software tools need to be written to make the job of using the chip easier. The first tool would aid in the task of programming the chip. Instead of programming the chip by hand, a user would write a specification of what events she would like to monitor in a special purpose language. A translator would then translate the user's specifications into the proper data file necessary to program the chip. The translator will hide any idiosyncrasies in the design, such as the requirement that the output code uniquely identify a single comparator row. In addition to the programming software, an output post-processor is needed to translate the raw output data from the monitor into data that is more suitable for human consumption.

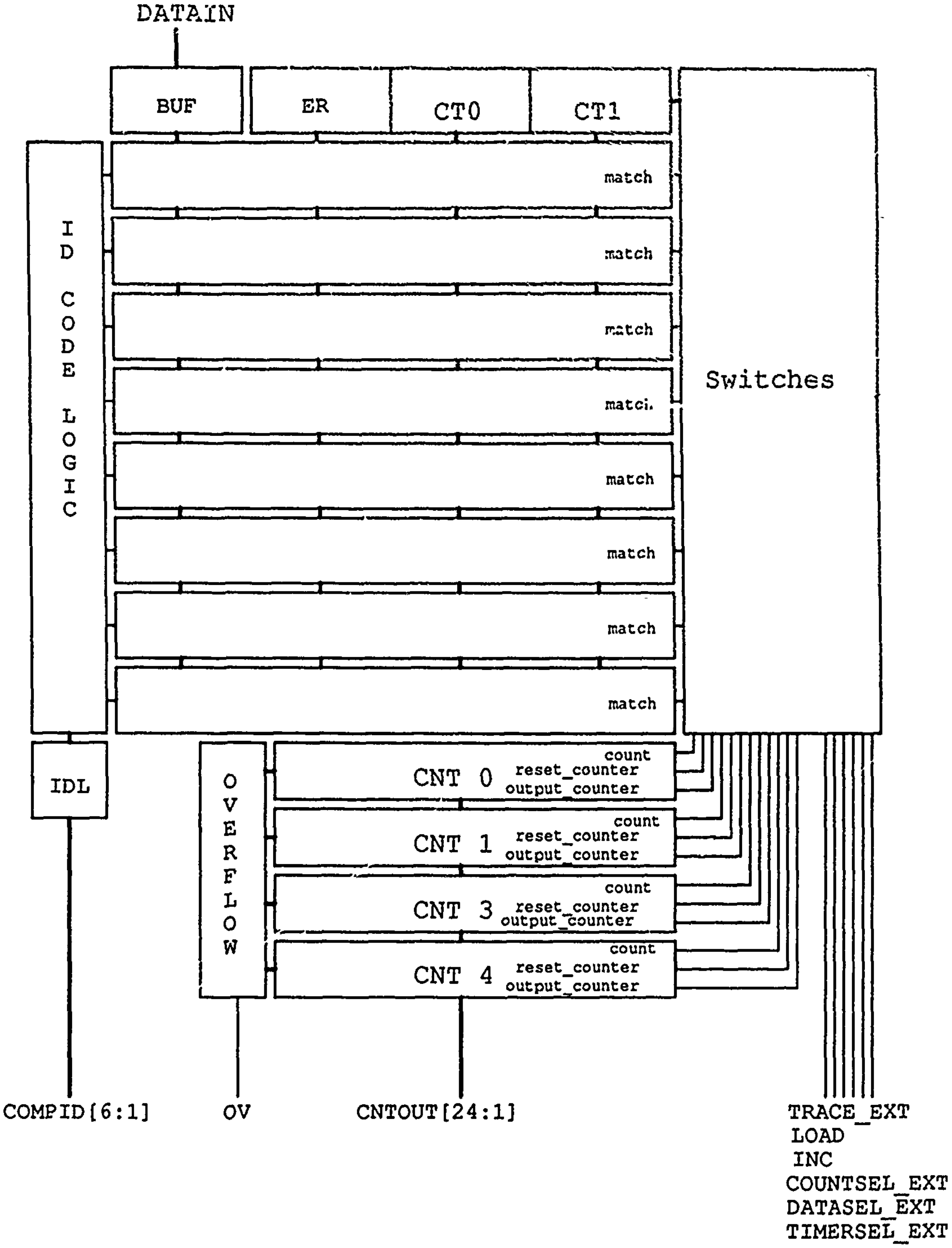
6 Acknowledgements

We thank the following participants in the graduate Advanced VLSI course, Winter 1989, for their contributions to the work here described: Tod Amon, Craig Anderson, Kathy Armstrong, Kevin Bolding, Prof. Gaetano Borriello, Suzanne Bunton, Tien-fu Chen, Ken Fox, Rick Hood, Ho-Shyan Lin, Brian Lockyear, Chi-Shung Wang, and Ken Whaley.

References

[Bunton 89] S. Bunton, G. Borriello, 1989. *A High Bandwidth Lempel-Ziv Data Compression Chip.*

DATA PATH



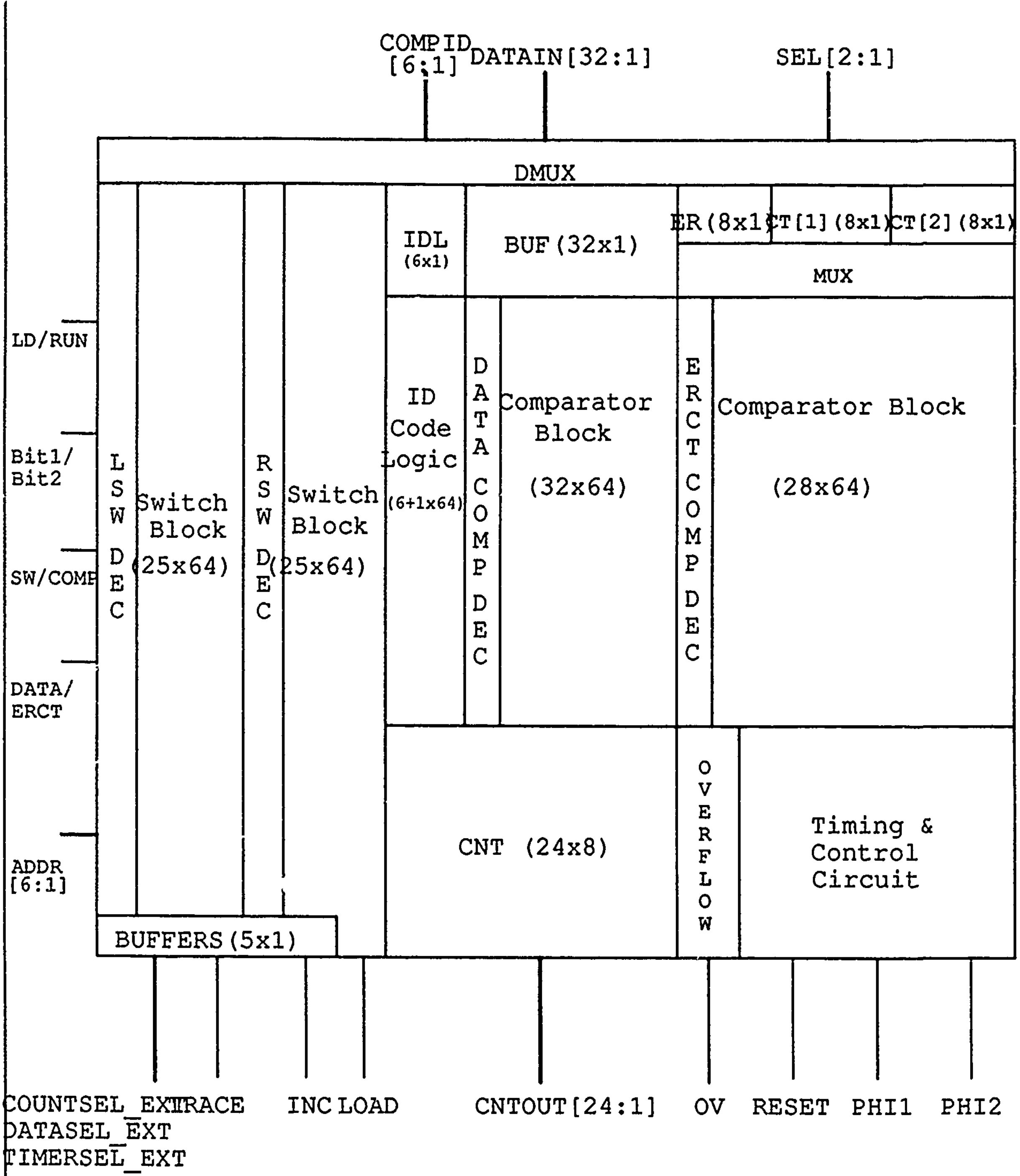
CS 568

Name: CS 568

Project: Trace Filter Design

Date: Winter Quarter 1989

FLOOR PLAN



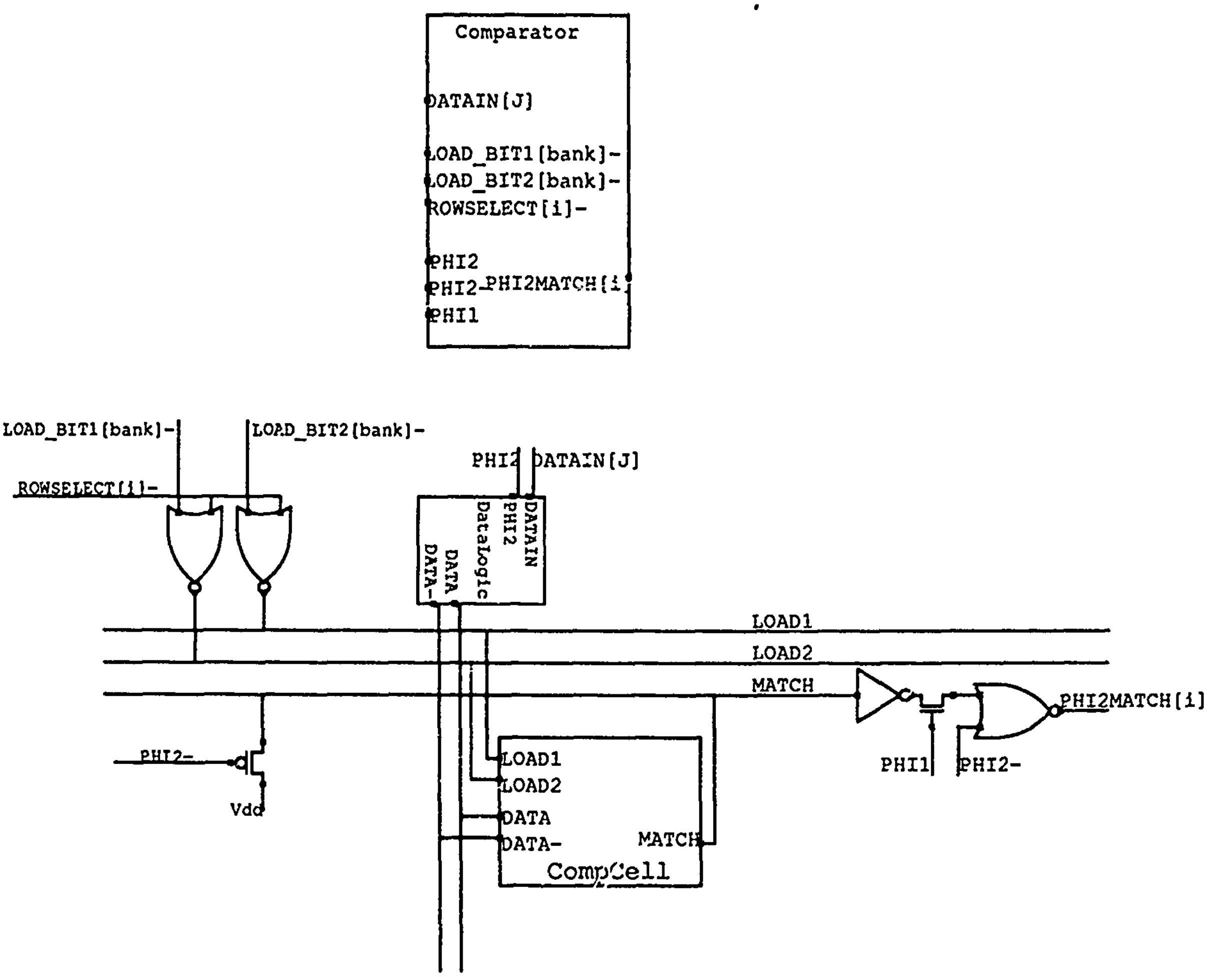
CS 568

Name: CS 568

Project: Trace Filter Design

Date: Winter Quarter, 1989

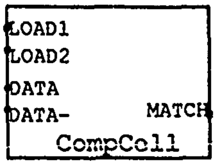
Comparator

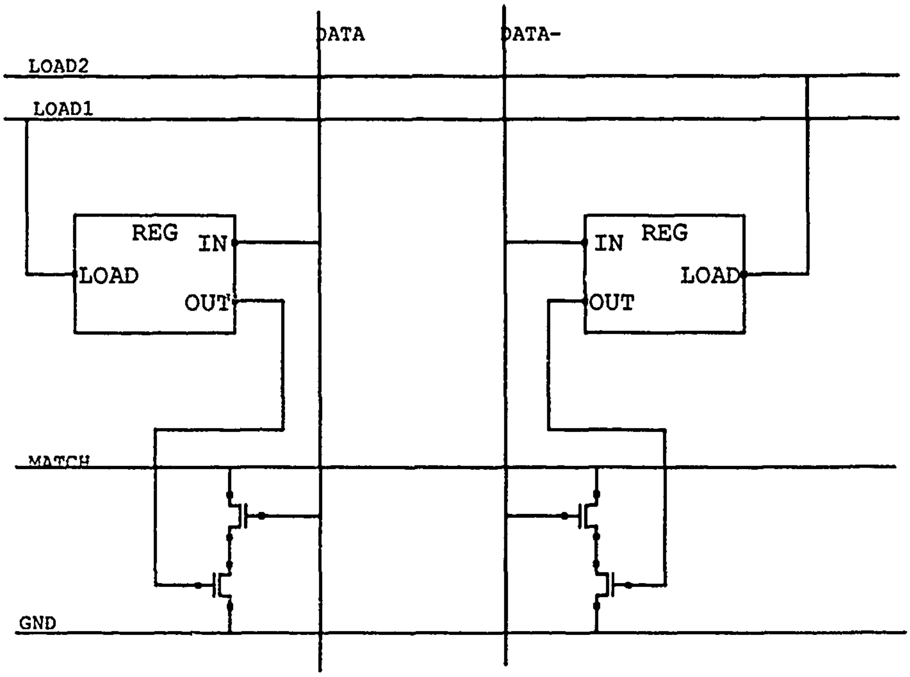


Notes: - The precharge transitor and the row select/load logic is repeated once for every 32-bit comparator group.
- The latch on the match line is repeated once for every 32-bit comparator group.
- The Data logic is repeated for each bit, but only once for the entire number of comparators.
- Pulldown time for the match line of a 32-bit comparator is approx. 24ns worst case.

CS 568	Name:	CS 568
	Project:	Filter Chip Leaf Cells
	Date:	Winter Quarter 1989

COMPARATOR CELL





Operation:

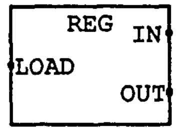
LOAD1	L AD2	DATA	DATA-	Effect
1	0	0	X	Set up to match 0's on DATA.
1	0	1	X	Set up to ignore value on DATA.
0	1	X	0	Set up to match 0's on DATA-.
0	1	X	1	Set up to ignore value on DATA-.
0	0	X	X	Match Line operates as set up above.

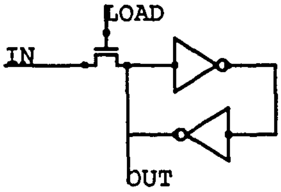
Notes:
- Match line must be precharged during phi2.
- During precharging, both DATA and DATA- must be 0 Volts.
- The comparator cell is 48h x 72w, taking into account shared power and gnd.

CS 568

Name: CS 568
Project: Filter Chip Leaf Cells
Date: Winter Quarter 1989

Register Cell



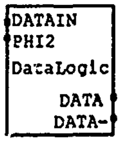


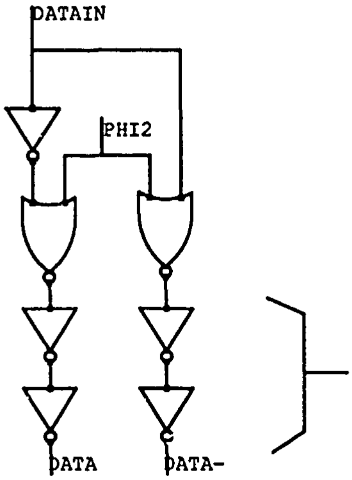
Notes:

- The pass gate must be large enough to overpower the lower inverter. This is especially crucial when writing a '1', since it is passed through an nmos pass gate.
- The inverter thresholds are adjusted to help alleviate the difficulty in writing a '1'. The upper inverter has a low threshold so a '1' may be written easily. The lower inverter has a high threshold so that it will output weak '0's and allow '1's to be more easily written over it's output.
- Load times are in the 3-4ns range. This assumes approx. 4.5V on the LOAD line and a good (within .5V of 0 or 5V) value on the IN line.
- The register comes in two flavors:
  - 32h x 30w (not including power and gnd). This is used in all applications except the ROM IDlatch.
  - 79h x 20w (including power and gnd). This is used for the ROM IDlatch, which needs to be skinny.

CS 568	
	Name: CS 568
	Project: Filter Chip Leaf Cells
	Date: Winter Quarter, 1989

Comparator Data Logic Cell





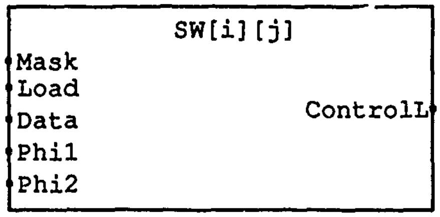
These inverters are sized to drive the large capacitance of the data lines.

Notes:
- This cell gates the data signals going into a single column of comparators. Its purpose is to ensure that the data and data- lines are always zero during the match line precharging on phi2.
- The cell is pitch-matched to the top of the comparator cell (72w) and is 107h.

CS 568

Name: CS 568
Project: Filter Chip Leaf Cells
Date: Winter Quarter, 1989

SWITCH CELL



i = 50:1
j = 64:1

Signals:

```
    Mask <-- DMUX <-- DATAIN pins (phi1)
    Load <-- LSW DEC or RSW DEC <-- DEC(phi1)
     Data <-- COMP (phi2)
   Control == CT_countL[k], CT_resetL[k] --> CT[k](phi1)
                                         --> MUX --> COMPs (phi1)
            == CNT_countL[k], CNT_resetL[k], CNT_enoutL[k} --> CNT[k](phi1)
             == ER_sseL[k], ER_resetL[k] --> ER[k](phi1)
             == INC_extL --> buffer --> INC output pin (phi1)
              == LOAD_extL --> LOAD output pin (phi1)
             == TRACE_extL --> buffer --> TRACE output pin (phi1)
             == DATASEL_extL --> buffer --> DATASEL output pin (phi1)
            == TIMERSEL_extL --> buffer --> TIMERSEL output pin (phi1)
            == COUNTERSEL_extL --> bufffer --> COUNTERSEL output pin (phi1)
```

Operations:

Normal operation:

Load	Mask	MR	Data	ControlL (assert low)
1	X	X	X	Load MR (See next table)
0	X	0	0	1
0	X	0	0	1
0	X	1	0	1
0	X	1	1	0

CS 568	
	Name: CS 568
	Project: Trace Filter Design
	Date: Winter Quarter, 1989

SWITCH CELL

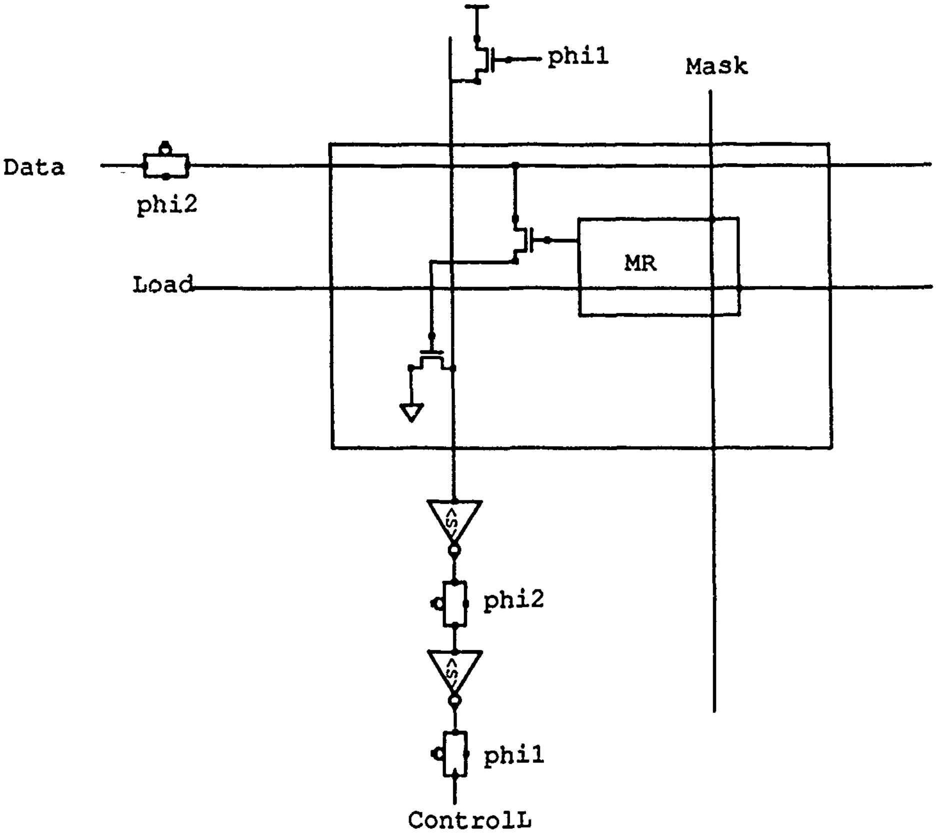
Load MR operation:

Load	Mask	Data	MR
0	X	X	normal operation
1	0	X	0
1	1	X	1

Size:

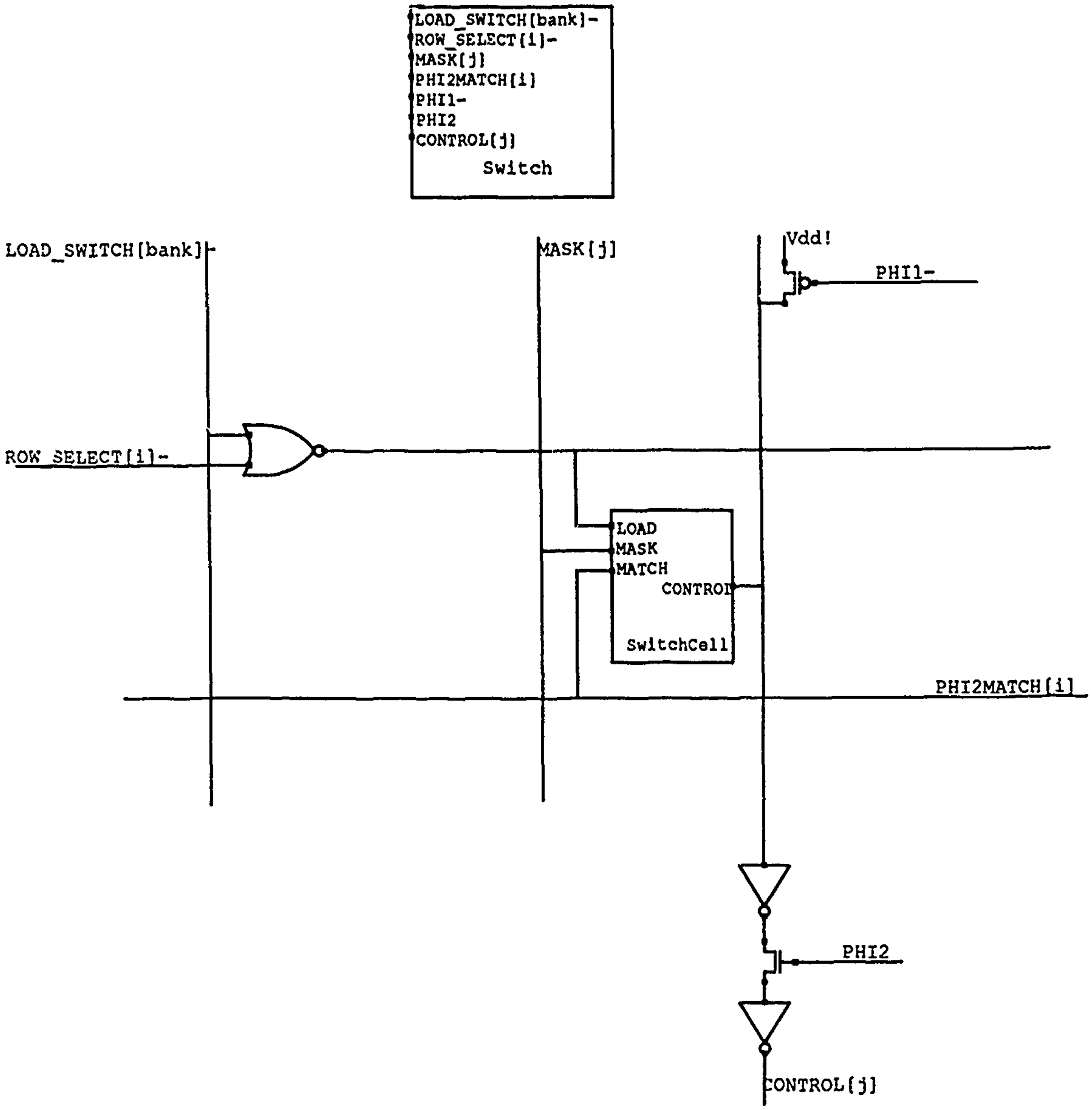
Height: same as COMP cells.

Implemention:



CS 568	Name: CS 568 Project: Trace Filter Design Date: Winter Quarter, 1989

Switch

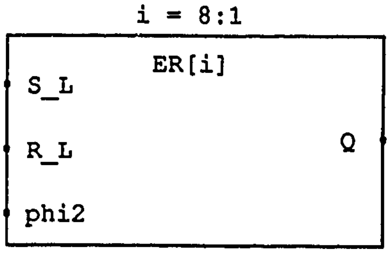


Notes: - PHI2MATCH must be low during precharging (phi1).
- Control line is latched in on phi2, available on phi1.
- The precharge and latch logic are shared for each control line.
- The row select logic is shared for each row of comparators in the same bank.
- Pulldown time for a 64-tall switch is approx. 13ns.

CS 568

Name: CS 568
Project: Filter Chip Leaf Cells
Date: Winter Quarter, 1989

EVENT RECORDER



Signals:

```
S_L <-- ER_setL[i] (during phi2)
R_L <-- ER_resetL[i] (during phi2)
 Q --> MUX --> COMP
```

Operations:

S_L	R_L	Q
1	1	Q(t-1)
1	0	0
0	1	1
0	0	x

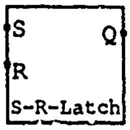
Size:

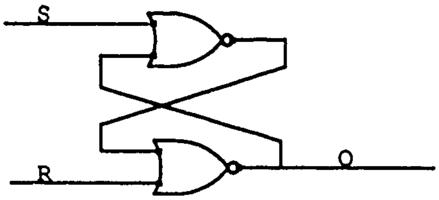
Width: same as COMP cell

CS 568

Name: CS 568
Project: Trace Filter Design
Date: Winter Quarter, 1989

Event Recorder Cell
(S-R Latch)

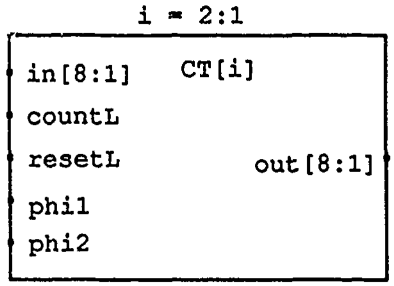




Notes: - The event recorder cell is a simple set-reset latch.
- The cell is 34h x 35w.

CS 568	Name:	CS 568
	Project:	Filter Chip Leaf Cells
	Date:	Winter Quarter, 1989

INTERNAL COUNTER/TIMER



Signals:

```
 in <-- hardwired 1
countL <-- CT_countL[i] from SW (phi1)
resetL <-- CT_resetL[i] from SW (phi1)
 out --> MUX --> COMP (phi2)
```

Operations:

```
If resetL then
   out := in
else
   if countL then
      increment( start at phi1; finished before the end
        of phi2 )
```

Size:

Width: same as COMP cell

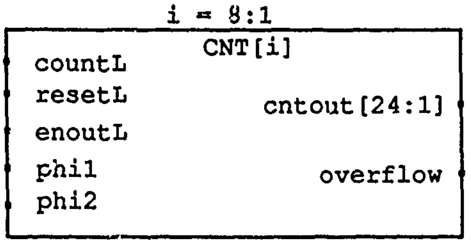
CS 568

Name: CS 568

Project: Trace Filter Design

Date: Winter Quarter 1989

COUNTER



Signals:

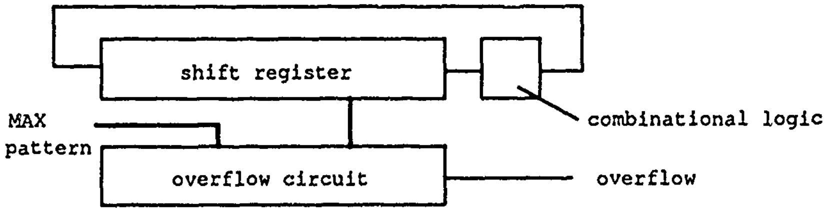
```
    countL <-- CNT_countL[i] from SW (phi1)
    resetL <-- CNT_resetL[i] from SW (phi1)
    enoutL <-- CNT_enoutL[i] from SW (phi1)
   cntout[24:1] --> CNTOUT[24:1] output pins (phi2)
    overflow --> OVERFLOW Circuit (phi1)
```

Operations:

```
If resetL == 1 then
     cntout := initial pattern
else if countL == 1 then
     cntout := next pattern
else if enoutL == 1 then
     output cntout

If cntout == MAX pattern then
     if countL(t+1) == 1 then
          overflow := 1
```

Implementation:



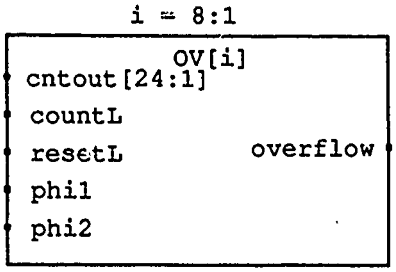
CS 568

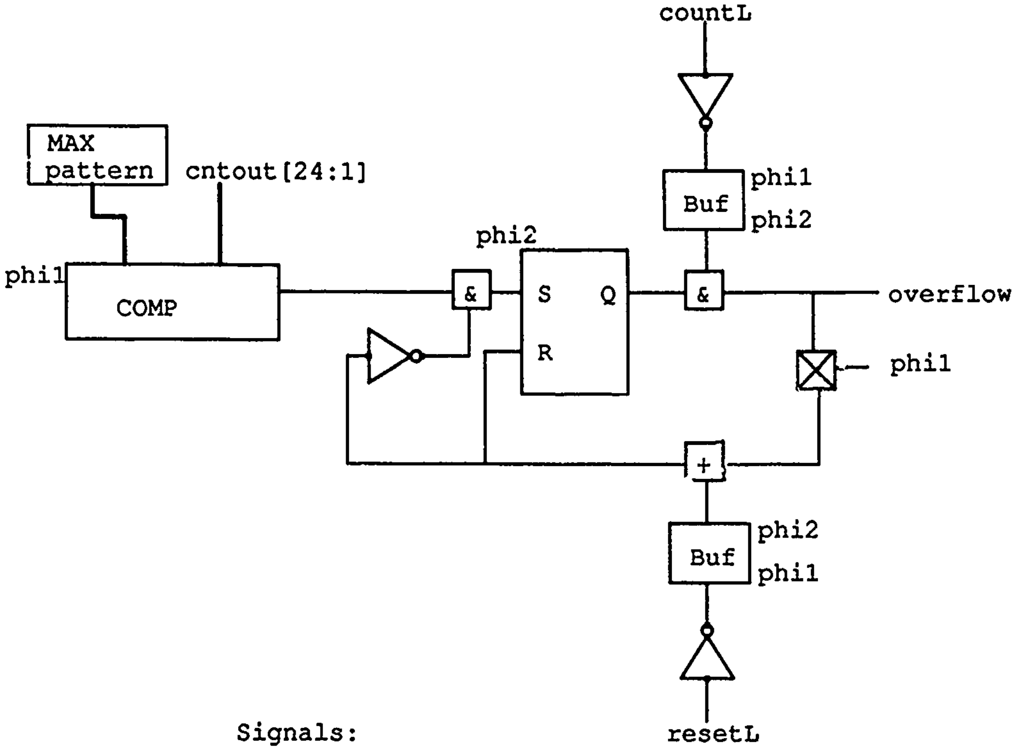
Name: CS 568

Project: Trace Filter Design

Date: Winter Quarter 1989

OVERFLOW CIRCUIT





Signals:

cntout <-- cntout from CNT[i] (phi1)
countL <-- CNT_countL[i] from SW (phi1)
resetL <-- CNT_resetL[i] from SW (phi1)
overflow --> OR gate of overflow signals from CNT[8:1]
--> OV output pin (phi2)

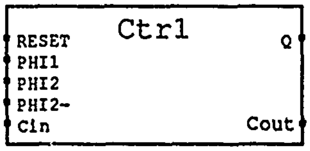
Operation:

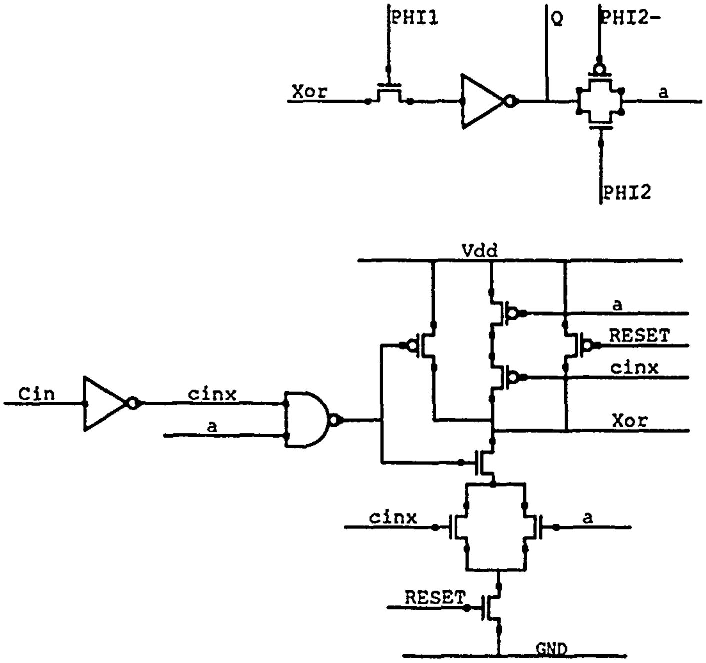
If (cntout == MAX pattern) and (countL occurs again before resetL) then
overflow := 1

CS 568

Name: CS 568
Project: Trace Filter Design
Date: Winter Quarter, 1989

Counter Cell



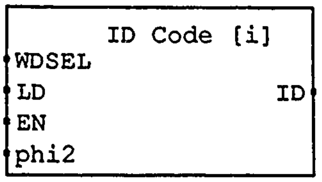


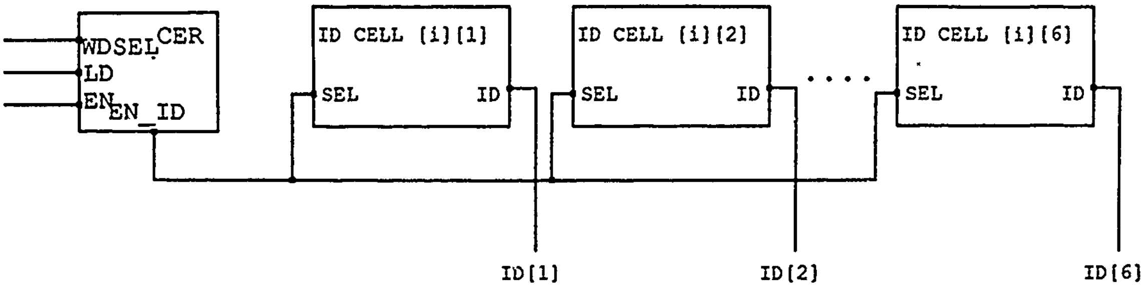
Note - The Cout to Carry-in signals are made through abutment.
The RESET signal sets this bit of the counter to zero.
The RESET signal is asserted low.
This cell is pitch-matched to the comparator cell. Its dimensions are 77h X 72w.

CS 568

Name: CS 568
Project: Filter Chip Leaf Cells
Date: Winter Quarter 1989

ID Code





Signals:

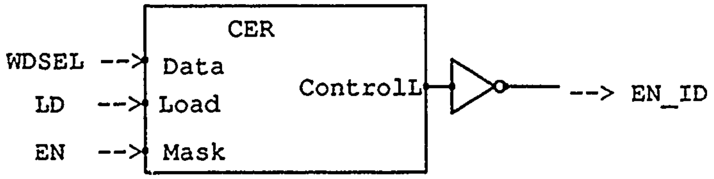
```
    LD <-- RSW DEC <-- DEC (phi1)
     EN <-- DMUX
    WDSEL <-- COMP_match (phi2)
     ID --> IDL (phi1)
```

Operations:

```
     if LD then CER := EN
    else if WDSEL = CER = 1 then
              EN_ID := 1
          else
              EN_ID := 0
```

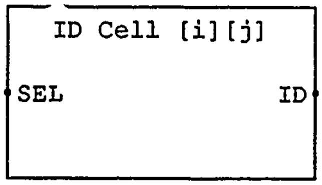
Implementation:

The CER (code enable register) can make use of the switch cell.



CS 568	Name: CS 568
	Project: Trace Filter Design
	Date: Winter Quarter, 1989

ID Code Cell



i = 64:1
j = 6:1

Signals:

```
SEL <-- EN_ID from CER (during phi2)
ID --> IDL latch --> COMPID output (phi1)
```

Operations:

```
if SEL = 1 then
   ID = hardwired pattern
```

Constraint:

There maybe more than one ID cells fired at the same time, The CER will generate a SEL signal only if the code should be output. The IDER (ID Code enable register) should be programmed so that no more than one code will be output simultaneously.

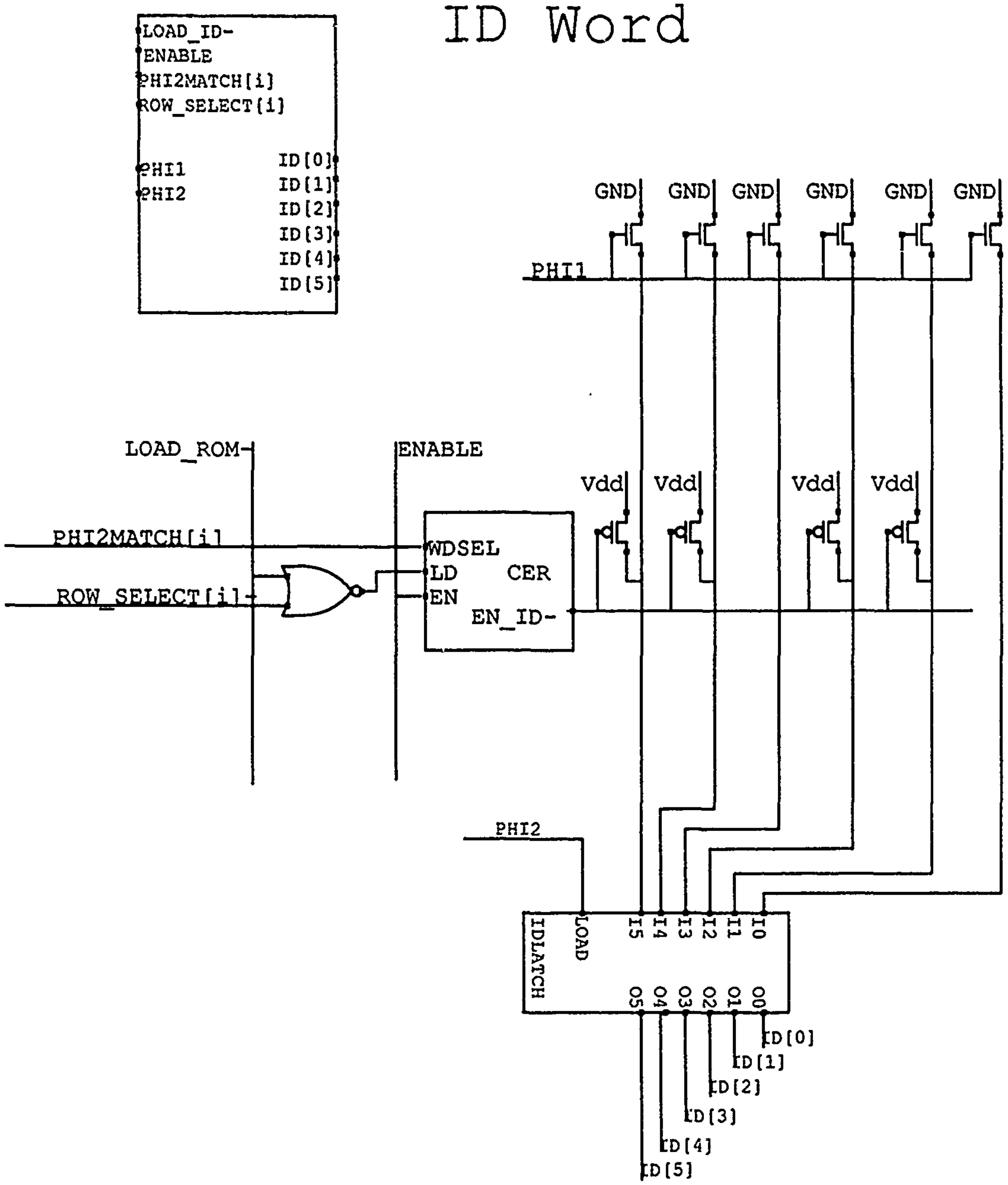
Size:

Height: same as COMP cells.

CS 568

Name: CS 568
Project: Trace Filter Design
Date: Winter Quarter, 1989

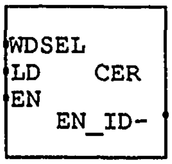
ID Word

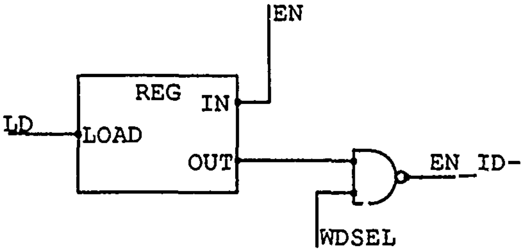


Notes: - This drawing represents a single code word (110110).
- ID lines are predischarged on PHI1 and evaluated on PHI2.
- PHI2MATCH[i] is the match line from comparator[i].
- Each code bit cell (pullup transistor) is pitch-matched to the comparator (48h) and is 12w. The 6-bit code word is 129w, including the cer.

CS 568	Name:	CS 568
	Project:	Filter Chip Leaf Cells
	Date:	Winter Quarter, 1989

Code Enable Register





Notes: - The ID code enable register outputs a zero on EN_ID- whenever it has been enabled and WDSEL is high.
- The cell is pitch-matched to the side of the comparator cell (48h) and is 55w.

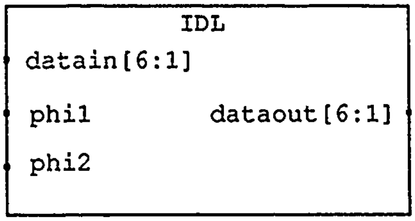
CS 568

Name: CS 568

Project: Filter Chip Leaf Cells

Date: Winter Quarter, 1989

ID LATCH



Signals:

```
datain[6:1] <-- ID from ID code logic (phi1)
dataout[6:1] --> COMPID output pins (phi2)
```

Operations:

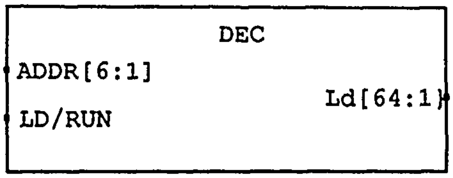
load datain on phi1
output on phi2

Size:

Width: same as ID code cell

CS 568	Name:	CS 568
	Project:	Trace Filter Design
	Date:	Winter Quarter, 1989

DECODER



Signals:

ADDR <-- ADDR input pins
LD/RUN <-- LD/RUN input pin

Ld --> LSW DEC (phi1)
RSW DEC (phi1)
DATA COMP DEC (phi1)
ERCT COMP DEC (phi1)

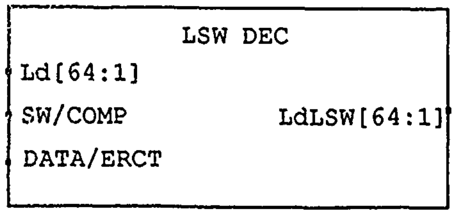
Operations:

Regular 6 to 64 decoder, enabled when Ld = 1.

CS 568

Name: CS 568
Project: Trace Filter Design
Date: Winter Quarter, 1989

DECODER



Signals:

```
  Ld <-- DEC
SW/COMP (switch/comp select) <-- SW/COMP input pin
DATA/ERCT (data/ER/CT) <-- DATA/ERCT input pin
LdLSW (load left half of SWs) --> SW_ldl lines (phil)
```

Operations:

```
LdLSW = (Ld = 1) * (SW/COMP = 1) * (DATA/ERCT = 1)
```

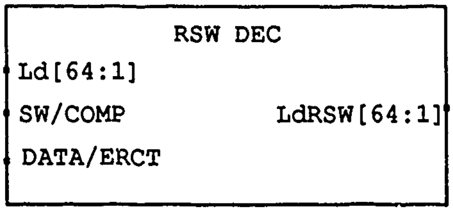
CS 568

Name: CS 568

Project: Trace Filter Design

Date: Winter Quarter, 1989

DECODER



Signals:

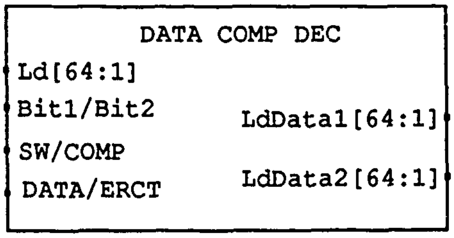
```
  Ld <-- DEC
SW/COMP (switch/comp select) <-- SW/COMP input pin
DATA/ERCT (data/ER/CT) <-- DATA/ERCT input pin
LdRSW (load right half of SWs) --> SW_ldr lines (phil)
```

Operations:

```
LdRSW = (Ld = 1) * (SW/COMP = 1) * (DATA/ERCT = 0)
```

CS 568	Name:	CS 568
	Project:	Trace Filter Design
	Date:	Winter Quarter, 1989

DECODER



Signals:

Ld <-- DEC
Bit1/Bit2 <-- Bit1/Bit2 pin
DATA/ERCT (select DATA or ERCT COMPs) <-- DATA/ERCT input pin
SW/COMP (switch/comp select) <-- SW/COMP input pin
LdData1 --> COMP's load Bit1 signals (phi1)
LdData2 --> COMP's load Bit2 signals (phi1)

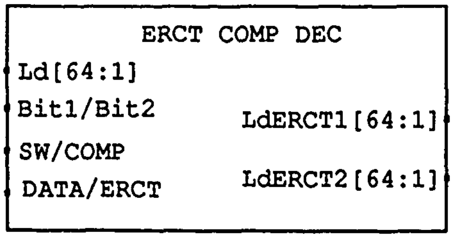
Operations:

LdData1 = (Ld = 1) * {SW/COMP = 0) * (DATA/ERCT = 1) * (Bit1/Bit2 = 1)

LdData2 = (Ld = 1) * (SW/COMP = 0) * (DATA/ERCT = 1) * (Bit1/Bit2 = 0)

CS 568	Name: CS 568 Project: Trace Filter Design Date: Winter Quarter, 1989

DECODER



Signals:

Ld <-- DEC
Bit1/Bit2 <-- Bit1/Bit2 pin
DATA/ERCT (select DATA or ERCT COMPs) <-- DATA/ERCT input pins
SW/COMP (switch/comp select) <-- SW/DATA input pin
LdERCT1 --> ERCT COMP's load Bit1 signals (phi1)
LdERCT2 --> ERCT COMP's load Bit2 signals (phi1)

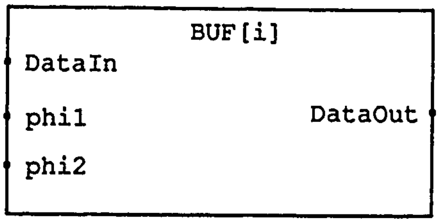
Operations:

LdERCT1 = (Ld = 1) * (SW/COMP = 0) * (DATA/ERCT = 0) * (Bit1/Bit2 = 1)

LdERCT2 = (Ld = 1) * (SW/COMP = 0) * (DATA/ERCT = 0) * (Bi:1/Bit2 = 0)

CS 568	Name: CS 568
	Project: Trace Filter Design
	Date: Winter Quarter, 1989

BUFFER



Signals:

DataIn <-- DMUX <-- DATAIN pins (phi2)

DataOut --> COMP (phi1)

Operations:

DataIn	DataOut
0	0
1	1

Size:

Width: same as COMP cell

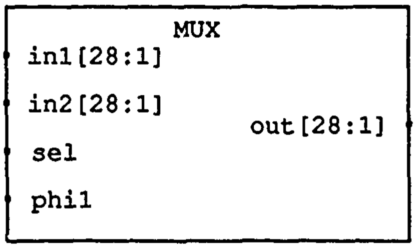
CS 568

Name: CS 568

Project: Trace Filter Design

Date: Winter Quarter 1989

MUX



Signals:

```
    sel <-- LD/RUN pin
   in1 <-- DMUX <-- DATA[28:1] pins
  in2 <-- ER[8:1] and CT[2:1][8:1] and CT_count[2:1] from SW
           and CT_reset[2:1] from SW (during phi2)
   out --> ER/CT COMPs (phi1)
```

Operation:

```
    case
     sel 0 : out := in2
     sel 1 : out := in1
    endcase
```

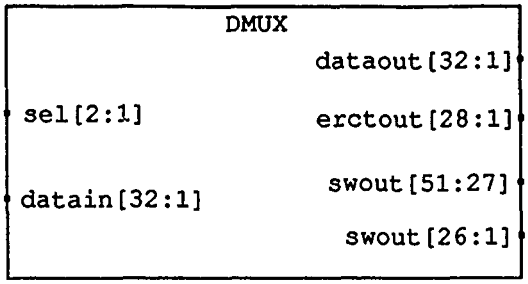
CS 568

Name: CS 568

Project: Trace Filter Design

Date: Winter Quarter, 1989

DMUX CELL



Signals:

```
sel <-- SEL[2:1] input pins
datain <-- DATAIN[32:1] input pins
dataout --> BUF --> data COMPs
erctout --> MUX --> ER/CT COMPs
swout --> SW and ID Code
```

Operations:

```
Case
 sel 0  :  dataout[32:1] := datain[32:1]
 sel 1  :  erctout[28:1] := datain[28:1]
 sel 2  :  swout[51:27] := datain[25:1]
 sel 3  :  swout[26:1] := datain[26:1]
Endcase
```

CS 568

Name: CS 568

Project: Trace Filter Design

Date: Winter Quarter, 1989